A Model-Driven Approach to Systems-of-Systems Engineering

Shayne Flint
Department of Computer Science
The Australian National University, Canberra, ACT 0200, Australia
Tel: +61 (0)2 61258183
shayne.flint@anu.edu.au

Abstract

Traditional engineering aims to solve problems by building, operating, maintaining and retiring systems of various kinds. However, problems usually emerge within dynamically complex environments of co-evolving technology, people, processes and other problems. Because of this, individual problems are often difficult to isolate and independently solve. It is generally accepted that these dynamically complex sets of interacting problems, or *Problem Situations*, are best dealt with by *improving* the whole rather than by attempting to *solve* individual problems. In recent years our research group has been thinking about the nature of engineering within such environments and has developed Aspect-Oriented Thinking (AOT): a novel model-driven engineering approach to continuous learning and improvement within dynamically complex environments. In this paper I provide an overview of AOT and show how it might form the foundation of an effective systems-of-systems engineering methodology.

1 Introduction

Traditional engineering aims to solve problems by building, operating, maintaining and retiring systems of various kinds. However, problems usually emerge within dynamically complex environments of co-evolving technology, people, processes and other problems (Ackoff 1999). Problems are often affected by the progress of other problems and changes throughout a problem situation. Because of this, individual problems are often difficult to isolate and independently solve. Attempts to do so often lead to the development and use of inappropriate systems.

It is generally accepted that dynamically complex sets of interacting problems, or *Problem Situations* (Checkland 1981), are best dealt with by continuously *improving* the whole rather than by attempting to *solve* individual problems (Ackoff 1999; Checkland 1981). In order to make appropriate improvements it is necessary to fully understand all aspects of the problem situation and the likely impact of any proposed improvements. To achieve such depth of understanding, the views and expertise of all applicable disciplines, including science, engineering, economics, psychology, philosophy, history and art, must be considered. It is particularly important that this be done in a way that respects the independence of each discipline and encourages, rather than weakens, their depth of knowledge and use of appropriate techniques, tools and processes.

In recent years our research group has been thinking about the nature of engineering within this context. In particular, we believe that new engineering methodologies should shift the *focus* of engineering away from the life-cycle of individual systems and towards a systematic process of *continuous learning and improvement* within problem situations. The scope of engineering methodologies should be expanded to include up-front activities that help people use appropriate multi-disciplinary knowledge, expertise and tools to learn about problem situations. These methodologies should then help people use the results of this learning to identify necessary improvements and make decisions regarding their implementation using appropriate physical, psychological, political, legal, financial, social and technological means.

In many ways, we are advocating a much broader multi-disciplinary vision of engineering.

To date, our work has resulted in the development of Aspect-Oriented Thinking (AOT) (Flint 2006; Flint 2008)[1]. This approach has a strong focus on the assembly of autonomous systems, capability, knowledge and other elements to form the systems required to both learn about and improve complex problem situations. The approach is currently being developed and evaluated within a number of different domains and reflects our research group's strong industrial background by supporting the actual implementation, operation and maintenance of real systems within contemporary industrial contexts.

Given the focus of AOT on problem situations and the assembly of independently evolving artefacts, it seems appropriate to explore the applicability of the approach within the context of Systems-of-Systems Engineering (SoSE) (Jamshidi 2005; Boardman and Sauser 2006). In the remainder of this paper, I provide an introduction to AOT within this context to show how it can form the foundation of an effective Systems-of-Systems Engineering methodology.

2 Overview

AOT is a continuous process of learning and improvement within complex problem situations. It represents a significant departure from traditional model-driven engineering approaches in which models are usually built as abstractions of a system of interest. As depicted in Figure 1, AOT models are not developed as abstractions of specific systems. Instead, *Domain Models* are used to describe autonomous and generalised *Domains* of knowledge, expertise and capabilities applicable to understanding a *Problem Situation* and making decisions regarding necessary improvements. Detailed *Specifications* for systems required to implement these improvements are developed in terms of, but separate from, these *Domain Models*. They are also formed in accordance with *Specification Archetypes* which describe the generic ways in which knowledge captured in *Domain Models* can be used to form

[1]While *Aspect-Oriented Thinking* shares the concept of cross-cutting concerns with *Aspect-Oriented Programming* (Elrad, Filman, and Bader 2002) within the software development domain, it has very little else in common.

Specifications for a class of systems. These *Specifications* can be often translated automatically to form the artefacts required to build, operate and maintain operational systems (eg. source code, build scripts and test data in the case of a software system).

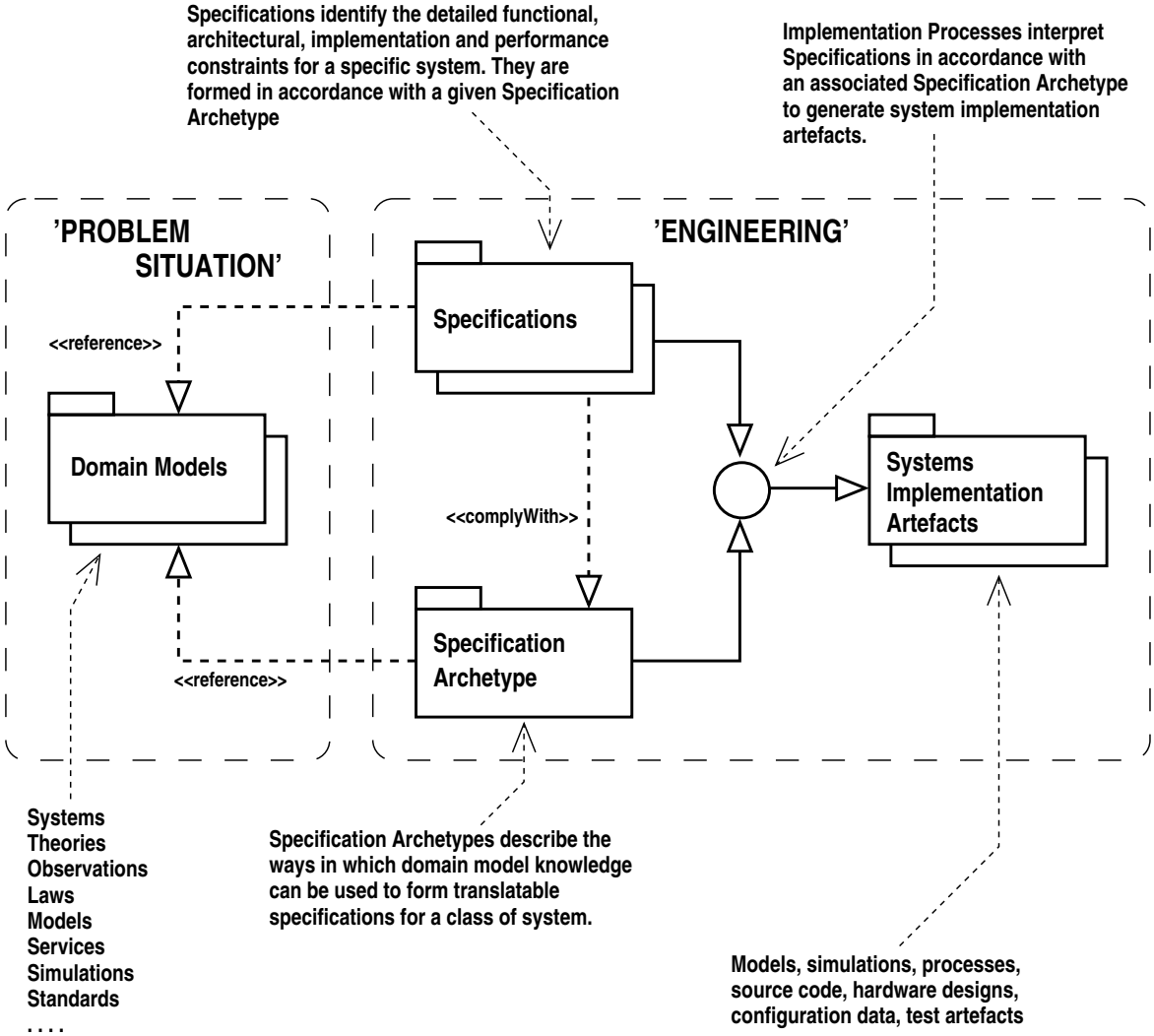

Figure 1. An overview of Aspect-Oriented Thinking and the flows of information involved. The diagram also highlights the separation of concerns between understanding a '*problem situation*' and '*engineering*' necessary improvements.

The next two sections are derived from previous work (Flint 2006; Flint 2008). Section 3 describes AOT concepts, while Section 4 describes how these concepts are used within a continuous process of learning and change based on Boyd's Observe-Orient-Decide-Act (OODA) loop (Boyd 1986).

3 Conceptual Model

Figure 2 depicts a Unified Modeling Language (Object Management Group 2006b) class diagram which identifies the concepts involved in AOT and the relationships

between them. The diagram is best read by forming sentences from class names, association phrases and multiplicities. For example, Figure 2 shows that a '*System*' – '*is built, operated and maintained to understand and/or improve*' – '*one or more*' – '*Problem Situations*', and that a '*Problem Situation*' – '*emerges from interaction among*' – '*one or more*' – '*Systems*'. By reading the diagram in this way, an overview of the concepts involved can be obtained. Details of each concept are described in the following sections.

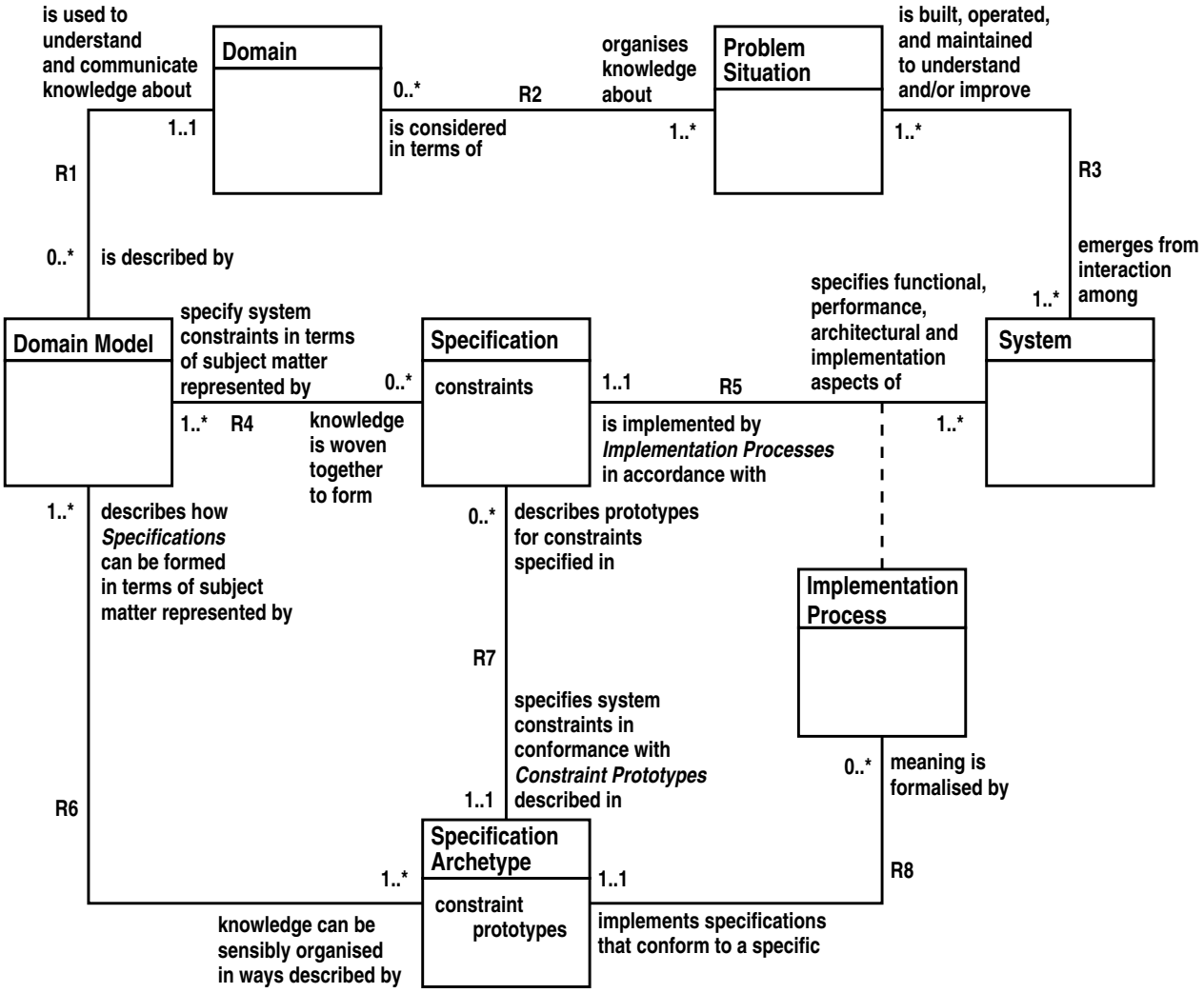

Figure 2. UML Class diagram representing the concepts involved in Aspect-Oriented Thinking.

3.1 Problem Situations and Systems

The overall objective of AOT is to make improvements within *Problem Situations*. For the purpose of this paper, a *Problem Situation* is an environment in which social, natural and man-made systems interact in ways that are considered unsatisfactory by one or more of the people involved.

Systems are viewed from two perspectives in AOT: the *emergence* and *improvement*

of *Problem Situations*. Problem Situations *emerge* from the interaction of social, man-made and natural systems. Social systems include humans both as individuals and organised into groupings such as teams and families. Natural systems include biological, chemical, geological and atmospheric structures. Man-made systems include concrete systems such as mechanical, electronic and urban systems as well as abstract systems such as plans, requirements and architectures.

Systems are also created, modified, operated and/or retired in order to *learn about and improve* a *Problem Situation*. This is achieved by applying concepts described in the remainder of this section in accordance with the AOT process described in Section 4.

3.2 Domains and Domain Models

In order to help people fully understand a *Problem Situation*, AOT applies the principle of separating concerns (Dijkstra 1982) in multiple dimensions. The various subject matters (*Domains*) involved in a *Problem Situation* are considered separately. For example, within the context of emergency services, *Domains* such as policing, fire-fighting and medical care, along with cross cutting *Domains* such as transportation, communications, and financing would be considered separately.

Knowledge about each *Domain* is captured in one or more autonomous *Domain Models* that each represents a particular view of a given *Domain* within a *Problem Situation*. Because they are autonomous, *Domain Models* can be independently developed and maintained by engineers, scientists, sociologists, psychologists, lawyers, philosophers, economists and others, using languages and techniques with which they are familiar. For example, within the broad context of systems-of-systems engineering, scientists could use traditional scientific methods and publications to '*model*' their observations, theories and experimental results, and business analysts could use the *Business Process Execution Language* (BPEL) (Organization for the Advancement of Structured Information Standards 2007) to describe organisational processes. Engineers could use languages and techniques such as *System Dynamics* (Forrester 1961), the *System Modeling Language* (SysML) (SysML Partners 2005), and the *Unified Modeling Language* (UML) (Object Management Group 2006b) to describe existing systems, and lawyers and politicians would probably use English to represent applicable laws and regulations.

Within any *Problem Situation*, there will be many '*models*' that have been developed with no knowledge of AOT and independently of any application of the approach. Examples include existing engineering artefacts, standards, research publications and laws. There are no limitations on using such models within the AOT approach. They are simply treated as *Domain Models* and are used as-is.

In summary, a *Domain* corresponds to knowledge and expertise related to a particular aspect of a *Problem Situation*. A *Domain Model* is a representation of *Domain* knowledge.

3.3 Specifications and Specification Archetypes

As outlined above, AOT represents a significant departure from traditional model-driven engineering approaches in which models are usually built as abstractions of a system of interest. AOT uses models for the specific purpose of communicating knowledge about autonomous aspects of a *Problem Situation*. *Specifications* that describe systems involved in a *Problem Situation*, as well as those required to learn about and later improve a *Problem Situation* are formed as sets of *Constraints* which are defined in terms of knowledge captured in *Domain Models*.

Each *Specifications* is also formed in accordance with a *Specification Archetype*, Each of these archetypes comprises an integrated set of *Constraint Prototypes*[2] which identify how elements from one or more *Domain Models* can be used to specify *Constraints* in a *Specification*.

For example, within the context of software development, a *Constraint Prototype* might describe how *Constraints* can be formed to state that instances of a *class* referenced in a UML class diagram, are to be made persistent using a specified relational database system. *Constraints* that conform to this prototype would refer to a specific *class* in one *Domain* and a specific *relational database* in another *Domain*. The meaning of this *Constraint Prototype* would be described in a natural language such as English. A more formal meaning would be provided by the actions of an *Implementation Process* (Section 3.4) which would generate code to create the appropriate database tables, and to read and write instance data.

More detailed examples of *Specifications*, *Specification Archetypes*, and how they are implemented in practice can be found in Flint 2006, Wang and Flint 2008 and Flint 2008.

Specification Archetypes will evolve over time into a set of *Constraint Prototypes* that is complete enough to support the formation of *Specifications* that fully describe systems that have a particular architecture and are implemented using specific technology. For example, within the software development domain, we might develop a *Specification Archetype* for the specification of web applications based on the *Linux, Apache, PHP and MySQL (LAMP)* architecture (O'Reilly 2008; Flint 2006). Such a *Specification Archetype* would be developed over time, starting with *Constraint Prototypes* that deal with data modelling and architectural concerns. Later, *Constraint Prototypes* would be added to deal with user interface and implementation concerns including HTML, the PHP programming language and Apache configuration. If the *Specification Archetype* needed to be later extended to support a Service-Oriented Architecture (Papazoglou and van den Heuvel 2007), *Constraint Prototypes* might be added to deal with the use of web services to implement activities described in a language such as the *Business Process Modeling Notation* (BPMN) (Object Management Group 2006a).

[2]The details of this concept have evolved since the original description of AOT (Flint 2006). Originally, it was called a *Requirement Archetype*. The concept, however, is not only about requirements. It is more about properties that are, or will be true, about a system. Hence, I now adopt the term *Constraint Prototype* and replace the concept of *Requirement* with *Constraint* to reflect this evolution.

Within the context of emergency services, a *Constraint Prototype* might deal with the way in which the implementation of an activity identified in a communications process can be implemented using a specific hardware item identified in a model of the service's communications network architecture. Another *Constraint Prototype*, might be developed to deal with using a particular aircraft type, operated by a specific organisation (a domain), to transport food and water described in yet another *Domain*.

3.4 Implementation Processes

Because of the relationship between *Specifications* and *Specification Archetypes* described above, it is possible to develop *Implementation Processes* which can process all *Specifications* that comply with a given *Specification Archetype*. That is, for each *Specification Archetype* there may be corresponding *Implementation Processes* which are able to implement aspects of all specifications formed in accordance with the given archetype. Specifically, *Implementation Processes* iterate through each *Constraint* stated in a given *Specification*. For each *Constraint*, the processes generate artefacts required to implement its intent as reflected in the description of the corresponding *Constraint Prototype*.

While *Implementation Processes* can be executed manually, automation can result in significant productivity gains over conventional approaches to system development.

4 Process Model

AOT is a *continuous process of learning and improvement* based on Boyd's *Observe-Orient-Decide-Act (OODA)* decision loop (Boyd 1986) as depicted in Figure 3. Within the context of improving a *Problem Situation*, information is collected and recorded during the *Observation* phase. During *Orientation*, this information is used to develop an understanding of a *Problem Situation* and to identify necessary improvements. Options regarding the development, operation, modification and retirement of systems required to *implement* these improvements are explored during the *Decision* phase. Selected options are then implemented during the *Act* phase. The loop repeats (now including the actual impact of earlier changes, as well as changes made to other parts of the problem situation) until all stakeholders agree that no further improvements are necessary.

This decision making model was chosen because of the insights it presents with respect to the dynamics and agility of change. A key concept underpinning the OODA loop is that decision makers who cycle through their OODA loop more rapidly than others, will have a competitive advantage. Slower decision makers make decisions based on observations and orientation that may be invalid by the time such decisions are implemented. As a result, their actions are often inappropriate and costly. More importantly, because the decisions of others may be more

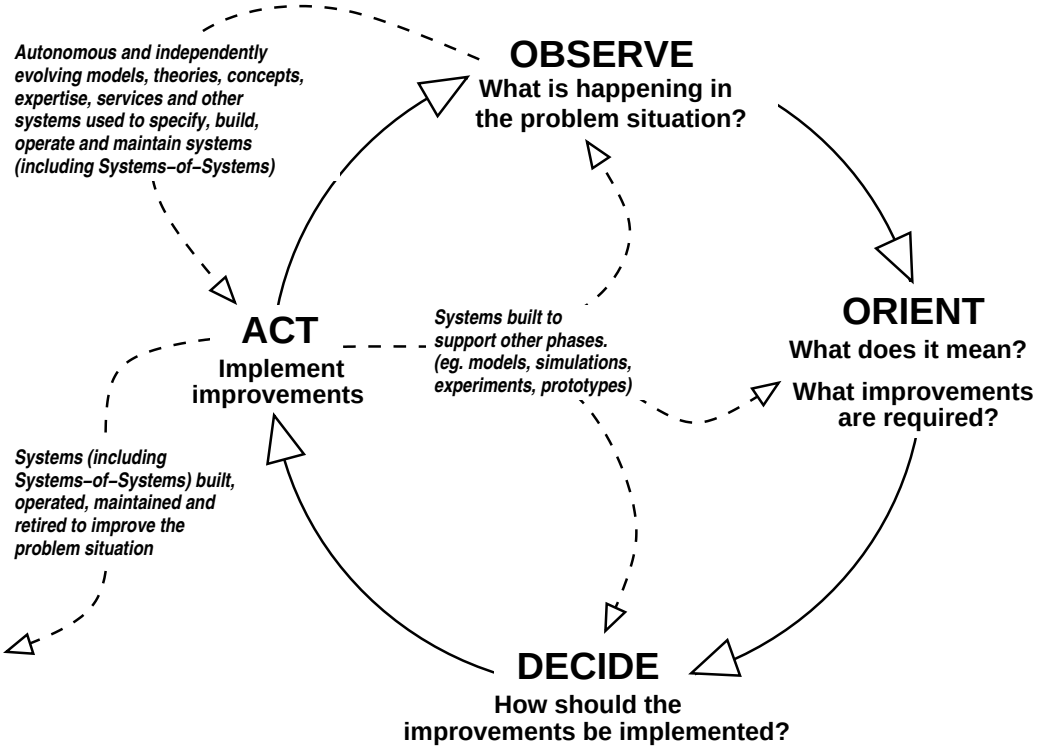

Figure 3. The process of Aspect-Oriented Thinking is based on Boyd's Observe-Orient-Decide-Act (OODA) loop. It aims to understand complex problems situations, identify necessary improvements and then implement them by building and operating systems of various kinds, including Systems-of-Systems (SoS).

agile, relentless and based on relevant and up-to-date observations and orientation, the world may appear chaotic to slower decision makers.

Note that the OODA loop described here will be operating at many places and at many levels within a single *Problem Situation*. They will be operating at different rates, under different sets of constraints and will be changing the *Problem Situation* in ways that impact the operation of other OODA loops. That is, they are themselves part of the dynamically complex *Problem Situation* they are being used to understand and improve.

The AOT approach recognises the importance of these ideas by adopting a model-driven approach to each phase of the OODA loop with a particular emphasis on reuse and automation where appropriate. The remainder of this section describes the process of AOT within the context of Systems-of-Systems Engineering.

4.1 Observation

During *Observation*, the various subject matters involved in a *Problem Situation* are identified as *Domains* (Section 3.2 and Figure 2). Autonomous *Domain Models* are then developed to represent a set of applicable views of each *Domain*. For example, separate *Domain Models* might be developed to represent structural, dy-

namic and aesthetic aspects of a mechanical device. Separate *Domain Models* might also be used to represent conflicting stakeholder views of a particular *Domain*. At this stage, such conflict is acceptable, as are incomplete models, because their purpose only relates to current understanding of a *Problem Situation* by the stakeholders involved. It is only when we need to use these models to form *Specifications* and *Specification Archetypes* that we would need to resolve (or ignore) the conflict enough and ensure that models are complete enough for the purposes of forming a *Specification* for a *particular* purpose.

Specifications and *Specification Archetypes* are also formed during *Observation* to learn about and communicate common patterns of organisation, interaction, behaviour and other aspects evident throughout a *Problem Situation*, and how these patterns manifest themselves in systems (and system-of-systems) that already exist within the *Problem Situation*.

4.2 Orientation

The aim of *Orientation* is to make sense of a *Problem Situation* and to identify necessary improvements. Understanding a *Problem Situation* is done without concern for the construction of any particular system and will involve understanding each *Domain* as well as the ways in which they are, and can be, woven together within a *Problem Situation*.

To understand each *Domain*, various traditional model analysis techniques can be used. For example, simulations and executable modeling languages such as $^{X}_{T}$UML (Mellor and Balcer 2002) and *System Dynamics* (Forrester 1961) may be used to understand the dynamics of a particular subject matter. That is, applicable *Domain Models* could be executed to ensure they correctly reflect stakeholder views and consensus.

In order to understand interactions between different *Domains*, *Specification Archetypes* and corresponding *Specifications* developed during the *Observation* phase can be analysed statically and/or dynamically. Static analysis involves traditional activities such as ensuring that the *Specifications*, *Specification Archetypes* and referenced *Domain Models* are properly formed and that they reflect acceptable levels of agreement between all the stakeholders involved. Dynamic analysis involves the use of simulations. If appropriate domain modeling languages such as $^{X}_{T}$UML (Mellor and Balcer 2002), *System Dynamics* (Forrester 1961) and the *Business Process Modeling Notation* (BPMN) (Object Management Group 2006a) are used, *Specifications* may be extended to include *Constraints* related to the implementation of simulations. These derived *Specifications*, could then be implemented and the resulting simulations used to explore the dynamics of systems involved in a *Problem Situation*.

Once the *Domains* and *Systems* involved in a *Problem Situation* are fully understood, stakeholders can work towards agreement regarding the identification of necessary improvements.

4.3 Decision

During the *Decision* phase, options for implementing improvements are evaluated using appropriate tools and techniques. Once evaluations are complete, *Specifications* for the systems required to implement the chosen options are formed.

While the identification of options is a human activity, AOT *Specifications* and *Specification Archetypes*, developed during the *Observation* and *Orientation* phases, can help by providing visibility of the ways in which existing systems are structured and behave.

Once options have been identified, *Specification Archetypes* and *Specifications* for systems required to implement each option can be formed. These *Specifications*, which are formed using reusable *Domain Models* developed during *Observation* and *Orientation*, can then be statically analysed or extended to form *Specifications* for simulations. These simulations can be used to explore the likely impact of each proposed implementation on the entire *Problem Situation* and to choose the most appropriate option for deployment.

After an implementation option is chosen, the necessary *Domain Models*, *Specification Archetypes* and *Specifications* are developed to fully describe the systems that must be built, operated, modified and/or retired to implement the chosen option. Again, this is done by reusing *Domain Models* developed during earlier phases, as well as *Domain Models* dealing with architecture and implementation concerns. In many cases, *Specifications* and *Specification Archetypes* built during earlier *Observation* and *Orientation* phases, will be reused by adding *Constraints* related to architecture, design, implementation, testing and deployment.

4.4 Action

The purpose of the *Action* phase is to implement decisions made during the *Decision* phase. This is achieved by developing and executing applicable *Implementation Processes*.

Note that the automation of *Implementation Processes* can significantly improve the agility with which an organisation can execute the entire AOT process.

5 Advantages

Advantages of the AOT approach include:

- **Multi-disciplinarity.** The separation of *Domain Models* from system specification supports multi-disciplinarity without the need for common languages, techniques and tools.

Domain Models can be developed in any appropriate language using any appropriate technique. AOT does not impose any constraints on the way knowledge is developed or represented. This allows each discipline to independently build their own bodies of knowledge, languages, techniques and expertise as they see fit and as they, in fact, do.

The concepts of *Specification* and *Specification Archetype* allow us to weave together this independently developing knowledge and expertise without weakening the depth of knowledge involved or looking for some lowest common denominator. That is, Aspect-Oriented Thinking facilitates multi-disciplinary research and development while protecting the depth of knowledge provided by each discipline and respecting their cultures.

- **Reuse.** AOT supports extensive reuse.

  Domain Models, *Specifications* and *Specification Archetypes* will evolve over time and are extensively reused throughout the AOT process. Initially, *Domain Models* will be used to develop a static understanding of a *Problem Situation*. These *Domain Models* will be reused to form *Specifications* and *Specification Archetypes* to describe more complex aspects of a *Problem Situation*. These *Specifications* and *Specification Archetypes* might then be reused and extended to include *Constraints* that deal with simulation. These simulations would then be used to learn more about the dynamics of a *Problem Situation* and to identify any necessary improvements. The same original *Specifications* and *Specification Archetypes* may be again reused and modified to reflect various options for implementing these necessary improvements. Finally, they would be modified to form *Specifications* for operational systems which will be deployed to improve a *Problem Situation*.

  An additional perspective on this, is that *Domain Models* and *Specification Archetypes* can be treated as *Intellectual Assets* (Mellor 2002) that have tangible value to an organisation due to their highly reusable nature.

- **Stakeholder Agreement.** AOT is tolerant of stakeholder disagreement regarding the meaning, completeness, correctness and content of *Domain Models*.

  Within a universal context, or even within the context of a particular *Problem Situation*, reaching consensus among people from a wide range of disciplines and perspectives regarding the meaning, completeness, correctness and content of *Domain Models*, is very unlikely. Fortunately, AOT does not require agreement within such broad contexts. Instead, the need for agreement lies at the level of forming *Specification Archetypes* and corresponding *Specifications*. The context at this level is much more specific and localised, and will involve fewer stakeholders.

  Reaching agreement for a specific purpose within these narrower contexts is feasible, whereas reaching such agreement within the context of an entire *Problem Situation* is not.

- **Preparedness.** The concept of *Specification Archetypes* has important consequences in relation to preparations made and training conducted to deal with possible future scenarios.

Because *Specification Archetypes* capture knowledge about the ways in which domains can be assembled for a class of purposes, they can be used to capture knowledge regarding a range of possible responses to future scenarios. For example, within the context of emergency services, *Specification Archetypes* could be developed for situations such as flooding. These archetypes would include *Constraint Prototypes* that describe how elements of domains such as water supply, sanitation, shelter, medical evacuation and communications can be assembled to form a response to a flood emergency. Appropriate *Implementation Processes* (manual and automated) would be developed and practiced to enhance preparedness so that when such an emergency arises, the response can be swift, systematic and effective.

- **Interoperability.** The AOT approach to *Domain Models*, *Specification Archetypes* and corresponding *Implementation Processes* can facilitate improved interoperability.

  If organisations develop and encourage the reuse of *Domain Models* that describe data structures and formats, and *Specification Archetypes* that include *Constraint Prototypes* that deal with interfaces, then systems specified using these items will exhibit improved interoperability with similarly specified systems.

  Such an approach is advantageous because agreement regarding the content of these *Domain Models* is not tied to any particular system. In addition, the use of models, through corresponding *Implementation Processes*, is relatively non-intrusive. That is, system specification, development approach, budgets and schedule can run independently of the interoperability related *Domain Models*, *Specification Archetypes* and *Implementation Processes* it might reuse.

- **Technology Adoption.** The separation of *Domain Models*, *Specifications*, *Specification Archetypes* and *Implementation Processes* supports the rapid adoption of new technology. If new technology relates only to the implementation of systems, then it can be adopted by modifying existing *Implementation Processes* and then re-generating existing systems or affected parts of such systems.

  If, on the other hand, new technology introduces the possibility of new capabilities, then *Specification Archetypes* and corresponding *Implementation Processes* would be revised to reflect the new capabilities. System *specifications* formed in accordance with these revised *Specification Archetypes*, could then be modified to take advantage of the new capabilities and implemented in the normal way.

- **Agility.** The automation of *Implementation Processes* along with the AOT approach to capturing and reusing knowledge in *Domain Models*, *Specification Archetypes* and *Specifications* , increases the agility with which the AOT process (OODA loop) can be executed. This in turn improves the responsiveness of an organisation to external change and disturbances (as detected during the *Observation* and *Orientation* phases of the AOT process).

- **Provenance.** With appropriate version control processes and annotation systems, *Domain Models*, *Specification Archetypes* and *Specifications* will, by their nature, record the history of decisions made during the life of a *Problem Situation*. *Specifications* formed during the *Orientation* phase will reflect the various experiments conducted to understand a problems situation, while those formed during the *Decision* phase will record the various options explored for implementing identified improvements, along with any related experimental data and results.

6 Limitations and Improvement Strategies

Current limitations of the AOT approach and strategies to deal with them, include:

- **Evaluation.** AOT is a new approach to engineering and has not yet been evaluated on a significant project.

  Our research group recognises the importance of doing this and has developed a strategy to run AOT projects in parallel with traditional projects. One such project is underway within the services science domain.

- **Tool Support.** There is no effective tool support for AOT.

  A prototype tool has been developed to support an earlier version of the approach, but it proved cumbersome and difficult to use, especially for non-technical users. However, some thought has recently gone into the development of a new generation of tool, but actual development has not yet commenced.

- **Communication.** While the concepts of AOT are relatively simple, we are having great difficulty in getting people to understand the approach.

  It is interesting to note that people with a computing background, seem to have particular difficulty in understanding AOT. We have started to discuss this as a group in an attempt to find better ways to communicate the AOT approach. Preliminary thoughts are that the difficulty might relate to the fundamental shift from building models as abstractions of systems, to building models for the purpose of understanding aspects of a *Problem Situation*.

- **Formality.** At this stage in the evolution of AOT, the meaning of a *Constraint* is described informally within the corresponding *Constraint Prototype* using natural language. While the more '*concrete*' meaning provided by the actions of *Implementation Processes* described in Section 3.4 is effective, it might be useful to explore the development and/or adoption of a formal constraint language and associated reasoning tools.

- **Implementation Processes.** The development of production quality *Implementation Processes* may present some difficulties.

  Experience with the development of *Implementation Processes* that generate web-based software systems has been positive. However, some concern has

emerged regarding the development of larger *Implementation Processes* for industrial use. We hope to address this issue by applying the AOT approach itself to the development of AOT artefacts including *Specification Archetypes* and *Implementation Processes*.

7 Conclusion and ongoing work

This paper has presented a summary of research underway to develop effective model-driven approaches to engineering within dynamically complex problem situations. I have described how Aspect-Oriented Thinking, a specific example of such an approach, might be used as a foundation for Systems-of-Systems Engineering.

Several projects are underway to further develop and evaluate the ideas presented in this paper. Within the computational science domain, AOT is being used to develop systematic approaches to scientific work-flow and model coupling within heterogeneous environments. This work shares many of the characteristics of Systems-of-Systems Engineering, especially as it relates to the assembly of existing independently evolving systems to form new capabilities.

AOT is also being evaluated as an approach to multi-disciplinary science using a case study involving the development of a novel approach to ecological simulation by an internationally dispersed team of ecologists, modelers, computer scientists, mathematicians and engineers (Gignoux 2008).

Finally, we are developing and evaluating AOT as an approach to services engineering (Wang and Flint 2008).

References

Ackoff, R. L. (1999). *Ackoff's Best, His Classic Writings on Management*. New York: John Wiley & Sons.

Boardman, J. and B. Sauser (2006). System of systems - the meaning of of. In *IEEE/SMC International Conference on System of Systems Engineering*, Los Angeles, CA, USA.

Boyd, J. R. (1986). Patterns of conflict (unpublished). viewed 19 June 2008, <http://www.d-n-i.net/boyd/pdf/poc.pdf>.

Checkland, P. (1981). *Systems Thinking, Systems Practice*. New York: J. Wiley.

Dijkstra, E. W. (1982). EWD 447: On the role of scientific thought. In *Selected writings on Computing: A Personal Perspective*, pp. 60–66. Springer-Verlag.

Elrad, T., R. E. Filman, and A. Bader (2002). Aspect-oriented programming: Introduction. *Communications of the ACM 44*(10), 29–32.

Flint, S. R. (2006, July). *Aspect-Oriented Thinking: An approach to bridging the disciplinary divides*. Ph. D. thesis, Australian National University, Canberra, Australia.

Flint, S. R. (2008). A New Paradigm for Model-Driven Development. To Be Published (available from the author).

Forrester, J. W. (1961). *Industrial Dynamics*. Cambridge, MA: The MIT Press.

Gignoux, J. (2008). 3Worlds Web Site. viewed 22 June 2008, <http://threeworlds.biologie.ens.fr/>.

Jamshidi, M. (2005). System-of-systems engineering - a definition. viewed 20 June 2008, <http://ieeesmc2005.unm.edu/SoSE_Defn.htm>.

Mellor, S. J. (2002). Make models be assets. *Communications of the ACM 45*(11), 76–78.

Mellor, S. J. and M. J. Balcer (2002). *Executable UML, A foundation for Model-Driven Architecture*. Indianapolis, IN: Addison-Wesley.

Object Management Group (2006a). Business Process Modeling Notation Specification, version 1.0. viewed 19 June 2008, <http://www.omg.org>.

Object Management Group (2006b). Unified Modeling Language: Infrastructure, version 2.0. viewed 19 June 2008, <http://www.omg.org>.

O'Reilly (2008). LAMP: The open source web platform. viewed 20 June 2008, <http://www.onlamp.com>.

Organization for the Advancement of Structured Information Standards (2007). Web Services Business Process Execution Language Version 2.0. viewed 19 June 2008, <http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.pdf>.

Papazoglou, M. P. and W. van den Heuvel (2007). Service oriented architectures: approaches, technologies and research issues. *The VLDB Journal 16*(3), 389–415.

SysML Partners (2005). Systems Modeling Language (SysML) Specification, version 1.0a. viewed 19 June 2008, <http://www.sysml.org>.

Wang, Y. and S. Flint (2008). Towards a Service Engineering Methodology. To be published (available from the authors).